Area Versus Detection Latency Trade-Offs in Self-Checking Memory Design

Omar Kebichi*, Yervant Zorian** , Michael Nicolaidis*

* Reliable Integrated Systems Group, TIMA / INPG, 46 avenue Félix Viallet
38031 Grenoble Cédex France.

** AT&T Bell Laboratories, Engineering Research Center, Princeton, NJ, 08540, USA

Abstract

With the increasing need for on-line reliability, today's electronic systems often require certain levels of self-checking. Depending on its application, the level of self-checking, i.e. the detection latency, of a given system is determined. Most of the known on-line testing schemes provide a fixed level of self-checking, hence do not allow flexibility in meeting the allowed detection latency and hardware overhead. This presents a new self-checking scheme for memories (RAMs, ROMs, etc.), which provides trade-off between hardware cost versus detection latency. The scheme takes the required detection latency and determines the codes to meet the system requirements. The paper also illustrates the flexibility of this scheme with certain implementation examples.

Keywords:*Design-for-Testability,Self-checking circuits, On-line BIST, Concurrent Testing of Memories*

I - Introduction

Today's electronic systems make extensive use of memories in different levels including main memory cards, Flash, ROM and cache RAM chips used as local memories, diverse single-port, multi-port and cache RAMs and ROMs embedded in microprocessors and ASICs, etc. As a matter of fact designing reliable memories is essential for designing reliable electronic systems. Self-checking design can be a good solution for achieving reliability since it could ensure concurrent error detection by means of moderate hardware overhead.

The general structure of self-checking circuits is given in figure 1. The outputs of the functional block are encoded and the checker is used to verify it.

The goal to be reached by self-checking circuits is often called Totally Self-Checking (TSC) goal, i.e. the first erroneous output of the functional block provokes an error indication on the checker outputs. To ensure this goal, Carter [CAR 68] has introduced the basic ideas and Anderson [AND 71] has defined the Totally Self-Checking (TSC) property for functional blocks and for checkers. Later, Smith and Metze [SMI 78] have defined the Strongly Fault Secure (SFS) circuits and [NIC 84] has defined the Strongly Code Disjoint (SCD) checkers, which are the largest classes of functional circuits and of checkers allowing to ensure the TSC goal.

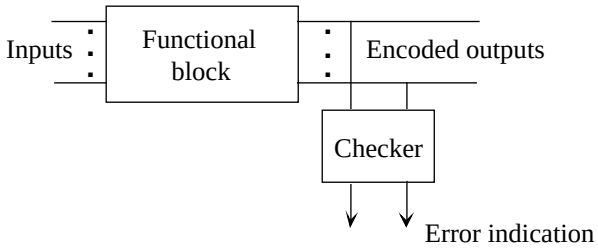

Figure 1 : General structure of self-checking circuits.

The TSC property means that the functional circuit is fault secure (each erroneous output due to a modelled fault is a noncode word), and self-testing (each modelled fault is detected by at least an input code word).

The SFS property means that the circuit is fault secure and not necessarily self-testing, but as long as undetectable faults occur the circuit remains fault secure.

Guaranting that the first erroneous output is detectable (TSC goal) offers a high level of safety but it could require a high hardware overhead. However, some applications can tolerate some errors to be undetectable as long as it is guaranteed that fault detection occurs within a short time after the first error occurrence. Then, one can take advantage of that in order to reduce hardware cost. This paper discusses hardware cost versus detection latency trade-offs in self-checking memory designs.

II - Self-checking memories

This section discusses the self-checking memory designs. The memory block diagram is given in figure 2. We remark that each cell of the memory cell array and each line of the multiplexer (MUX) is connected with only one output. This allows to ensure the SFS property for faults affecting these parts, by using a single parity bit (see for instance [NIC 87]). The extended use of the parity code for designing self-checking memories is based on this fact. However, the outputs of the decoder are connected with several memory outputs. Thus a single fault in the decoder can produce multiple errors on the memory outputs which are not detectable by the parity code. Thus, the existing self-checking memory designs do not cover the decoder faults. This is inconsistent with the objective of designing reliable electronic systems since even if the decoders represent a small part of the memory array, the reliability of the system can be

reduced considerably. Let us suppose for instance that in some memory the decoders represent 10 % of the memory area. If the MTBF for the whole memory is 10^{-5} faults per hour and if we use a self-checking scheme which does not cover 10^{-4} of the real faults, the level of safety will be10^{-9} undetectable faults per hour. On the other hand, if only the memory word array is checked the level of safety will be $10^{-1} \times 10^{-5} + (9 \times 10^{-1}) \times 10^{-5} \times 10^{-4} \approx 10^{-6}$ undetectable faults per hour, and therefore it is reduced by three orders. As a matter of fact, the appropriate checking of decoders is of great importance.
In the following we propose a technique which checks the decoder outputs after they have crossed over the memory cell array. This technique correlates hardware cost and detection latency.

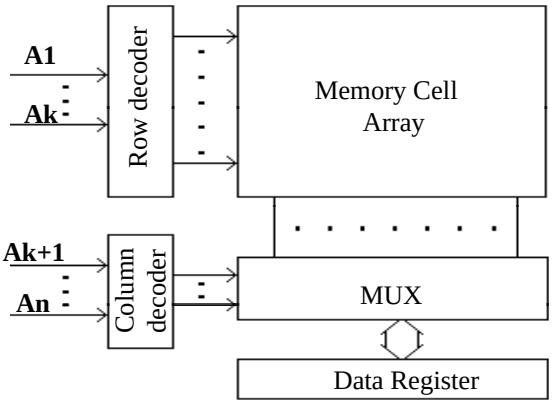

Figure 2 : Memory block diagram

III- Self-checking decoders

A previous scheme for self-checking decoders [NIC 94] uses a ROM matrix that receives the outputs of the decoder and generates an unordered code having a number of code words higher than or equal to the decoder outputs. This scheme guarantees the strongly fault secure property for all stuck-at faults regardless to the design of the decoder.
Recall that an unordered code is a code such that no code word covers another code word. That is no code word has 1's in all positions another code word has 1's.
An implementation of this scheme generates a Berger code with information bits equal to the decoder inputs. Recall that a Berger code is a code having check bits that count the number of 0's in the information bits. Another implementation of this scheme generates an m-out-of-n code (each m-bit code word has exactly m 1's) with $C_m^n \geq N$ (N being the number of the decoder outputs).
Another self-checking decoder scheme [CHE 85], [NIC 84.b] uses a ROM matrix that receives the outputs of the decoder and generates even and odd parity of the decoder inputs. For a decoder implementation that uses a n-input AND or NOR gate per decoder output, this scheme covers the majority of faults. However, for a multilevel implementation (e.g. several levels of 2-input gates) the scheme results on low fault coverage and large detection latency.
The new scheme presented in this paper will check the decoders by using a NOR matrix (ROM matrix) having as inputs the outputs of the decoder and generating an unordered code. The number of code words of this code will be chosen to meet the tolerated detection latency.
If the number of code words is equal to the outputs of the decoder, then we have the scheme proposed in [NIC 94] that requires the higher hardware cost and offers zero detection latency (the first erroneous output is detected).
If the number of code words is 2 (1-out-of-2 code), then we have the scheme proposed in [CHE 85], [NIC 84.b] which requires the lower hardware cost but result on the highest detection latency.

Behind the selection of unordered codes there are the following reasons :
- Consider the stuck-at 0 fault of a decoder output. When an error is produced due to this fault, then, no decoder output is selected. Thus, the output of the NOR-gate matrix is the all 1's vector which cannot belong to an unordered code. The detection latency is therefore zero.

- Consider the stuck-at one fault of a decoder output. Then, any error due to this fault selects two outputs of the decoder. If the code words generated by these two outputs are different, the output of the NOR matrix is guaranteed to be a noncode word and the error is detected. This is because for bit positions that take different values in the two code words, the NOR matrix output will be 0. The resulting word is covered by both code words and do not belong to the unordered code. This does not hold if the code is not unordered. Of course, if the code words generated by the two selected outputs of the decoder are the same, the error is not detected whatever is the code.
Similar considerations hold for the stuck-at faults in internal nodes of the decoder.

III.1 Preliminary code mapping construction

If the number of code words of the unordered code is lower than the outputs of the decoder then, it is impossible to detect all errors produced by the stuck-at 1 faults. This is because, in that case, there are at least two decoder outputs generating the same code word. Thus, when both of these lines are selected due to a stuck-at 1 fault in one of them, the error is not detected. Therefore 0 latency cannot be guaranteed.
In that case, for a given number of code word, the efficiency of the scheme depends on the way the code words are mapped. Consider for instance a decoder with n inputs (a_1,a_2,...,a_n) and $N = 2^n$ outputs (A_1,A_2,..., A_N).
Consider that the unordered code has 2^{n-k} code words. Such a code can be implemented by using a NOR matrix with $(n-k)+\lceil \log_2(n-k) \rceil$ outputs. The n-k outputs generate

the values of the signal $a_1,a_2,...,a_{n-k}$ and the remaining $\lceil \log_2(n-k) \rceil$ outputs generate the Berger code check bits for these signals. Then, any fault on a part of the decoder that decodes some of the inputs $a_{n-k+1},a_{n-k+2},...,a_n$ never produces a detectable error and the detection latency for these faults is infinite. To avoid this situation the code must be constructed to guarantee a uniform (or close to uniform) distribution of detection latency over the decoder faults. The following construction will satisfy this constraint.

Firstly we will consider the q-out-of-r codes with $q = \lceil r/2 \rceil$ (or $q = \lfloor r/2 \rfloor$) because they are the unordered codes that require the minimum number of bits for a given number of code words. These codes have a number $a = C_q^r$ code words.

Let us associate, with each value $0 \leq B < a$, a code word of the q-out-of-r code. And, let us then associate with each address the value B = A.mod(a) where A is the arithmetic value of the address. We have $0 \leq B < a$. This way a code word of the q-out-of-r code is associated to each address and thus to each decoder output. The decoder will be checked by implementing a NOR matrix (ROM) that encodes the decoder outputs into the above code words. This scheme is shown in figure 3. In the final construction the value of a will be modified slightly for reasons explained in the next section.

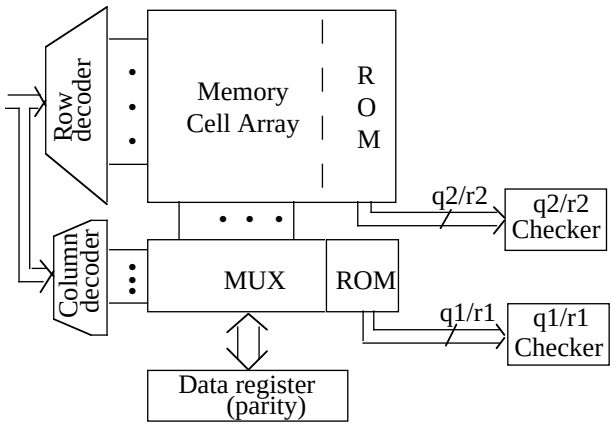

Figure 3 : Self-check ing Memory.

III. 2 Detection latency computation and final code mapping

To compute the detection latency we need to describe the decoder in a structured manner. The decoder will be implemented by using one or more levels of t-input AND gates.

Each address value will be decoded by a tree of AND gates that generates one output of the decoder. These trees have common terms. Thus, after minimization (factorization) several AND gates will be shared by several trees. The resulting multilevel circuit can be described as follows:

Each level of the circuit is divided into several blocks, and each of these blocks decodes the values applied on a subset of the decoder inputs. The set of these decoding blocks is partitioned into t-tuples (since t-input gates are used). Then, in the next level a decoding block is used for each t-tuple. This new block combines the outputs of the blocks of the t-tuple to decode the union of the sets of inputs decoded by the blocks of the t-tuple.

The computation of the detection latency will be done by considering that the decoder is implemented by using trees of 2-input AND gates. This way the computation will be valid if gates with more inputs are used, since in such a case we need to consider the stuck-at faults for only a subset of the nodes of the 2-input gates network. The different levels of this implementation are :

• 0-level : For each input $a_1,a_2,...,a_n$ of the decoder a decoding block, composed of one inverter, is used to obtain the direct and complementary values of the input. For instance, the decoding block A_1 produces a_1 and $\overline{a_1}$, the decoding block A_2 produces a_2 and $\overline{a_2}$, etc.

• 1-level : The decoding blocks of 0-level are associated into pairs (A_1,A_2), (A_3,A_4), (A_5,A_6),... having distinct elements. For each pair (A_i, A_j), a decoding block A_{ij} composed of 2-input gates is used. Each gate receives an input from the block A_i and an input from the block A_j. The block A_{ij} decodes the inputs i and j. Each output of A_i is combined with both the outputs of A_j. Thus four 2-inputs gates are used in A_{ij}.

• k-level : Let $i = 2^{k-1}$. The decoding blocks of k-1 level are associated into pairs $(A_{123...i}, A_{(i+1)(i+2)....(2i)})$, $(A_{(2i+1)(2i+2)....(3i)}, A_{(3i+1)(3i+2)....(4i)})$... . For each of these pairs a decoding block, composed of 2-input gates, is used. Each gate receives an input from the first block of the pair and a second input from the second block of the pair. Each output of the one block is combined with all the outputs of the other block of the pair. Thus 2^{2i} 2-input gates are used in each block of k-level.

• Last-level : When we reach a level having two decoding blocks then, only one pair can be formed and the next level will have exactly one decoding block. This level is the last level and generates the outputs of the decoder.

The structure described above is valid if n is a power of 2. In this structure the pairs used in a level k are formed by decoding blocks belonging to the level k-1, and each decoding block of level k decodes the same number ($j = 2^k$) of inputs of the decoder. If n is not a power of 2, then, some pairs used in some levels will be formed by decoding blocks belonging to two different levels, and the decoding blocks of such a level do not decode the same number of decoder inputs. The analysis presented below is valid regardless to the value of n.

Two important properties hold.
a- in the fault-free decoder each decoding block of any level has exactly one output equal to 1.

b- if a fault forces the outputs of a decoding block to the all 0's state, the outputs of the decoder will be in the all 0's state.

Let us now compute the detection probability of a fault (stuck-at faults are considered in this analysis).

- Stuck-at 0 faults :

Consider the stuck-at 0 of the output of a gate belonging to the decoding block $A_{(j+1)(j+2)....(j+i-1)}$ of level k. This block decodes the i inputs a_{j+1}, a_{j+2}, ..., a_{j+i-1} of the decoder. The stuck-at 0 line decodes a particular value on these inputs. An error will be produced when this particular value is applied on these input lines. This error will be of the type $1 \rightarrow 0$. In that case from property **a-** all the outputs of the decoding block will be 0 and from property **b-** all the outputs of the decoder will be 0. Then, all the outputs of the NOR matrix will be 1 and the error is detected since this word cannot belong to an unordered code. Therefore, for any stuck-at 0 fault we have zero detection latency.

- Stuck-at 1 faults :

Consider a stuck-at 1 fault on the same line as the stuck-at 0 considered above. This fault produces an error if the value applied on a_{j+1}, a_{j+2}, ..., a_{j+i-1} is different than the value decoded by the line affected by the fault. The error is of the type $0 \rightarrow 1$. Then, from property **a-**two outputs of the block $A_{(j+1)(j+2)....(j+i-1)}$ are equal to 1. Then, in the (k+1)-level, these active outputs will be combined with the active output of another decoding block (the block paired with $A_{(j+1)(j+2)....(j+i-1)}$) and will produce two active outputs in the (k+1)-level. This is repeated until reaching the last level which will have again two active outputs. Thus, exactly two outputs (say L1, L2) of the decoder will be equal to 1. The address decoded by these two outputs differ only on the signals a_{j+1}, a_{j+2}, ..., a_{j+i-1}. Then, if A_1, A_2 are the arithmetic values of the addresses decoded by L1 and L2, we will have A_1-A_2 = m_1-m_2 (where m_1 is the arithmetic value of the signals a_{j+1}, a_{j+2}, ..., a_{j+i-1} that select the stuck-at 1 line and m_2 is the arithmetic value actually applied on these signals). If $m_1.\mathrm{mod}(a) \neq m_2.\mathrm{mod}(a)$, the error is detected. Thus, the probability of detecting the fault within a single clock cycle is given by the probability of having $m_1.\mathrm{mod}(a) \neq m_2.\mathrm{mod}(a)$.

For the block that decodes the i inputs a_0, a_1, ... a_{i-1}, the arithmetic values (i.e. m) that are applied on the decoded inputs are 0,1,2,....,2^i-1. The function m.mod(a) maps these values into the a values 0,1,2,...,a-1. Thus the probability of having $m_1.\mathrm{mod}(a) \neq m_2.\mathrm{mod}(a)$ will be close to 1/a. For a block that decodes the i inputs a_j, a_{j+1},...,a_{j+i-1} the arithmetic values applied on the decoded inputs are equal to 2^j.X where X takes the values 0,1,2,...,2^i-1. If the higher common divider of 2^j and a is 1, then, the function m.mod(a) will map these signals into the values 0,1,2,...,a-1 and the detection probability is again close to 1/a. However, if the higher common divider of 2^j and a is f > 1, then, the function m.mod(a) will map the values of 2^j.X into the b values 0, f, 2f, ...,(b-1)f with b = a/f. In this case the detection probability will be close to 1/b and is detected by about f times. Thus, a must be selected not to have common dividers with 2^j. In other terms a must be odd. For the codes q-out-of-r having odd C_q^r, a will be selected equal to C_q^r. For the codes with even C_q^r, a will be selected equal to C_q^r - 1.

Finally, in the particular case of the 1-out-of-2 code we will replace the mapping based on the mod(a) function by the mapping that generates the odd and even parities of the decoder inputs. This mapping avoids the above problem and a can be selected equal to C_1^2 = 2. With this mapping all errors produced by stuck-at one faults in a block decoding i inputs with $2^i \leq a$ will be detected since, in this case, $m_1 \neq m_2$ will imply $m_1.\mathrm{mod}(a) \neq m_2.\mathrm{mod}(a)$. Thus, the detection latency is 0.

For blocks with $2^i > a$ the lower value of the probability of non detecting the faults within a single clock cycle is $\lceil 2^i / a \rceil / 2^i$. This probability is close to 1/a but, its exact value depends on i. Its minimal value corresponds to the blocks that decode the lower number of inputs i satisfying the relation $2^i > a$.

The probability of not detecting the fault within c clock cycles is $P_{ndc} = (\lceil 2^i / a \rceil / 2^i)^c$. Thus when the required values of c and P_{ndc} are known a can be found by replacing successively the values i = 1,2,3,4, ...etc. until to find the first value of a satisfying $2^i > a$. If the value of a find as above is even, this value is increased by 1. Then, we select the code q-out-of-r with minimum r that satisfies the relations $C_q^r \geq a$ and $q = \lceil r/2 \rceil$ (or $q = \lfloor r/2 \rfloor$). The final value of a used in the function B = A.mod(a) in order to determine the code mapping is C_q^r if C_q^r is odd, or C_q^r - 1 if C_q^r is even.

When a is selected equal to C_q^r - 1, one code word of the q-out-of-r code is never generated on the outputs of the NOR matrix. In order to make the q-out-of-r code complete (for the purposes of exercising of the m-out-of-n checker), one address mapped to some other code word can be mapped to this code word.

For instance, if we need to detect the faults within c = 10 clock cycles with an escape probability $P_{ndc} = 10^{-9}$ or less we find a = 8 and the code satisfying $C_q^r \geq 8 + 1$ is the 3-out-of-5 code having $C_q^r = 10$. The value of a used in the function B = A.mod(a) will be 10 - 1 = 9.

IV- Implementation and hardware overhead

Let q1-out-of-r1 be the code used for the column decoder and q2-out-of-r2 the one used for the row decoder. Consider a RAM with m-bit words, which has a row decoder with p inputs (2^p outputs) and a column decoder with s inputs (2^s outputs) (n = p + s address lines). The area overhead required to check the decoders for the scheme of figure 3 is dominated by the area of the two ROMs. This overhead is $k(r_1.2^s + r_2.2^p) / m.2^n$, where k is the ratio (ROM cell width) / (Memory cell width). As an example for a RAM having 1K words of 16 bits and a 1-out-of-8 column multiplexing, considering k = 0.3 and using the 3-out-of-5 code for both decoders, the area overhead will be 1.9%. The area overhead introduced by the two 3-out-of-5 code checkers is insignificant. The overhead required for the parity checking of the data is: 6.25% (= 1/16) for the parity bit and 0.15% for the parity checker, resulting on a total area overhead of 8.3%.

The self-checking scheme presented in this paper has been evaluated for the case of embedded RAMs in the AT&T Microelectronics 0.4µm CMOS standard cell library. Three sizes of embedded RAMs are used: 2K words by 16 bits, 4K words by 32 bits, and 8K words by 64 bits. In Table (1) the required escape probability is set to $P_{ndc} = 10^{-9}$ and a range of values for c (2 to 40 clock cycles) is evaluated. For each c, the corresponding q-out-of-r code is derived and the added hardware is measured by the percentage of hardware increase to the real area of RAM (width×length in μm^2). The wide range of hardware increase in Table (1) demonstrates the flexibility of the scheme and the trade-off between the hardware cost versus detection latency.

C	q-out-of-r code	% of hardware increase 16×2K	32×4K	64×8K
2	9-out-of-18	88.7	49.35	26.28
5	5-out-of-9	44.35	24.6	13.14
10	3-out-of-5	24.8	13.7	7.3
20	2-out-of-4	19.5	9.67	5.84
30	2-out-of-3	15	8.2	4.38
40	1-out-of-2	9.7	5.48	2.92

Table (1)

In Table (2), c is fixed to 10 clock cycles and a range of P_{ndc} (10^{-2} to 10^{-30}) is covered. Similar to Table (1) the three sizes of RAMs and their percentage of hardware increase is demonstrated.

Pndc	q-out-of-r code	% of hardware increase 16×2K	32×4K	64×8K
10^{-2}	1-out-of-2	9.7	5.4	2.92
10^{-5}	2-out-of-4	19.5	9.6	5.84
10^{-9}	3-out-of-5	24.8	13.7	7.3
10^{-15}	4-out-of-7	34.2	19.1	10.2
10^{-20}	5-out-of-9	44.35	24.67	13.14
10^{-30}	7-out-of-13	63.5	35.6	18.9

Table (2)

Similar trade-offs can be obtained if the self-checking scheme is implemented on memory types other then RAMs, such as ROMs, CAMs, etc.

V- Conclusion

A cost-efficient self-checking scheme for memories is presented. The scheme uses parity coding for the data to ensure zero detection latency for single-cell faults. As for decoder faults, a technique allowing hardware cost versus detection latency trade-offs is introduced. This technique starts from the detection latency allowed by a given application and determines the decoder scheme that meets this requirement using the lowest hardware cost.

References

[AND 71] ANDERSON D.A "Design of Self-Checking Digital Networks Using Coding Techniques" . Urbana, CSL Uni. of Illinois, Sep. 1971 (rep. 527).

[CAR 68] CARTER W.C., SCHNEIDER P.R. "Design of Dynamically Checked Computers", IFIP Congress, Edinburgh 1968, Inf. Proc. 68, Amsterdam, NL, 1969.

[CHE 85] CHEN C. Y. , FUCHS W. K., ABRAHAM J. A., "Efficient Concurrent Error Detection in PLAs and ROMs" Proc. IEEE Int. Conf. on Comp. Design, Port Chester, NY. USA, Oct. 1985.

[NIC 84] NICOLAIDIS M., JANSCH I., COURTOIS B. "Strongly Code Disjoint Checkers" 14th FTCS, Kissemmee, USA, 1984.

[NIC 84b] NICOLAIDIS M. "Antémémoire : Version Autotestable", Document N°.2, Projet SCQM, THOMSON, Oct. 1984.

[NIC 87] NICOLAIDIS M. "Shorts in Self-Checking Circuits" Proc. IEEE Int. Test Conf., Washington D.C., Sep. 1987.

[NIC 94] NICOLAIDIS M. "Efficient UBIST for RAMs" 12th IEEE VLSI Test Symp., Apr. 1994, Cherry Hill, USA.

[SMI 78] SMITH J.E. METZE G. "Strongly Fault Secure Logic Networks" IEEE Trans. on Comp., Vol. C-27, N°.6, June 1978.